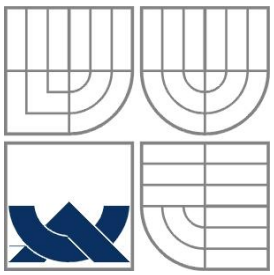
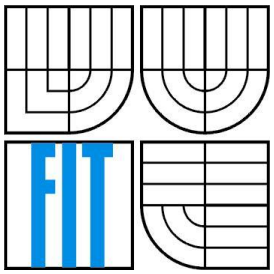


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

ČÁSTICOVÉ EFEKTY V POČÍTAČOVÝCH HRÁCH

PARTICLE SYSTEMS IN COMPUTER GAMES

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

MICHAL ZACHARIÁŠ

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. TOMÁŠ MIKOLOV

BRNO 2009

Abstrakt

Cílem této bakalářské práce je čtenáři přiblížit problematiku částicových systémů, jejich historii a současný trend, který stále více směřuje k využití akcelerace pomocí GPU grafických adaptérů. Dále se zabývá návrhem a implementací částicového systému tak, aby byl minimalizován počet výpočetních operací nutných pro simulaci chování částic.

Abstract

Aim of this bachelor thesis is to explain issue of particle systems, their history and today's trend, which more and more tends to acceleration using GPU of graphical adapters, to the reader. Furthermore it describes design and implementation of particle system to minimize number of computational operations needed to simulate behavior of particles.

Klíčová slova

počítačová grafika, částicový systém, historie částicových systémů, implementace částicového systému, částice, billboarding

Keywords

Computer graphics, particle system, history of particle systems, implementation of particle system, particle, billboarding

Citace

Michal Zachariáš: Částicové efekty v počítačových hrách, bakalářská práce, Brno, FIT VUT v Brně, 2009

Částicové efekty v počítačových hrách

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Tomáše Mikolova. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Michal Zachariáš
10.5.2009

Poděkování

Děkuji vedoucímu práce panu Ing. Tomáši Mikolovovi za jeho cenné rady a připomínky.

Obsah

1 Úvod

Tato bakalářská práce se zabývá využitím částicových systémů jako grafického prostředku, pro tvorbu efektů v moderních počítačových hrách, návrhem co nejméně časově a výpočetně náročného řešení pro jejich zobrazení, dále pak vytvořením, implementací a optimalizací vhodných datových struktur, abstraktních datových typů a programového kódu.

Pojem grafika vychází z řeckého *grafein*, což znamená kreslit. Tento pojem je velmi těžko definovatelný, ale obecně můžeme říct, že se jedná o druh výtvarného umění, kde autor za pomoci jedné z grafických technik vytvoří umělecké dílo, které rozmnoží určitým řemeslným postupem [1].

S rozvojem informačních technologií vzniká pojem *Počítačová grafika*, což je z technického hlediska obor informatiky, který používá počítače k vytváření a vykreslování umělých grafických objektů [2]. Vývoj počítačové grafiky umožnil počítačům jednodušší a rychlejší manipulaci a také interpretaci různých typů grafických dat. Zapříčinil revoluci ve filmovém průmyslu, do kterého přinesl možnost vytvářet velice realistické speciální efekty, které do té doby nebylo možné technicky realizovat.

Herní průmysl byl samozřejmě rozvojem počítačové grafiky zasažen také. Vývojářům umožnil vytvářet hry s daleko propracovanějšími efekty, realističtějším prostředím, osvětlením a fyzikou. Pro velké množství efektů (např. výstřel zbraně, oheň, kouř za raketami, exploze, tekoucí voda, unikající palivo…) v počítačových hrách jsou využívány různé typy částicových systémů.

2 Základní pojmy

V této kapitole objasníme některé ze základních pojmů počítačové grafiky a pojmů z oblasti částicových systémů, které se v textu později objeví.

2.1 Částicový systém

Částicový systém umožňuje simulovat a vykreslovat jevy, které je složité, ne-li nemožné, reprodukovat pomocí obvyklých simulačních a vykreslovacích metod. Mezi tyto jevy patří například oheň, kouř, mraky, exploze, tekoucí voda, vlasy, tráva a některé abstraktní grafické efekty jako je magie a technologie budoucnosti (teleportace, efekty energetických zbraní…). Jedná se tedy o jevy, které nelze přesně definovány nějakou sadou primitiv (např. trojúhelníky nebo polygony) a nemající pevný tvar.

Konkrétně se jedná o systém hmotných bodů (částic) pohybujících se ve dvourozměrném (2D) nebo trojrozměrném (3D) prostoru, který modeluje daný jev. Jejich počáteční vlastnosti bývají náhodné a pohyb je ovlivněn působením lokálních a globálních vlivů.

2.1.1 Základní model částicového systému

Pro každý rámec (*frame*) animace jsou vykonány tyto kroky:[3]

1. Vygenerujeme nové částice
2. Každé nové částici jsou přiřazeny její počáteční vlastnosti
3. Všechny částice, které přesáhly svou dobu existence, jsou odstraněny ze simulace
4. Zbylé částice jsou aktualizovány (posun, rotace, změna barvy, změna průhlednosti, změna velikosti, detekce kolize, aplikace globálních a lokálních vlivů…)
5. Částice jsou vykresleny

Vytváření částic a určování jejich vlastností se může řídit různými principy (např. z vědy, strojírenství…). V této práci použijeme nejobvyklejší způsob - náhodné generování.

2.1.2 Náhodné generování částic

Existují dva způsoby, kterými lze vytvořit náhodný počet částic. U prvního programátor určuje, kolik částic se vygeneruje za jeden rámec animace:

$$počet = průměrnýPočet + R \cdot variacePočtu, \quad \text{kde } R \in \langle -1,1 \rangle \tag{2.1}$$

Druhou metodu lze použít, pokud je vyžadována kontrola úrovně detailů (*LOD = Level Of Details*). U této metody je počet vygenerovaných částic závislý na obsahu plochy obrazovky, kterou částice pokrývají. Tedy například: čím dále jsme od částic, tím méně obsahu pokrývají a je jich generováno méně, ale efekt zůstává vizuálně stejný.

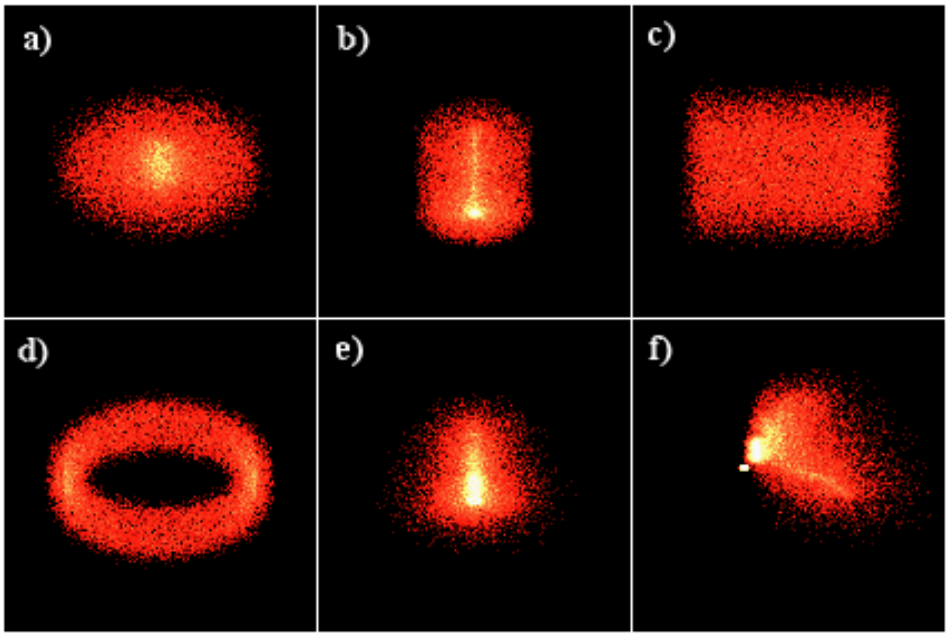
$$počet = (průměrnýPočetNaPixel + R \cdot variacePočtuNaPixel) \cdot obsahPlochy, \quad \text{kde } R \in \langle -1; 1 \rangle \tag{2.2}$$

2.2 Emitor částic

Je speciální objekt, který je zdrojem částic. Jeho poloha a rotace ve dvourozměrném nebo trojrozměrném prostoru udává pozici, kde jsou částice generovány, a směr, kterým budou pokračovat.

2.2.1 Tvary emitorů

Emitory mohou mít tvar různých geometrických útvarů (viz obrázek 2.1, obrázek převzat z [4]). Nejběžnější bývá bod, ale můžeme se setkat i se čtvercem, obdélníkem nebo kruhem či elipsou. Tvar ovšem není omezen na dvourozměrné útvary. Krychle, koule nebo dokonce nepravidelný objekt (*mesh* - vytvořený jedním z profesionálních programů pro práci s trojrozměrnou grafikou např. 3D Studio MAX) mohou být použity jako emitory částic.



Obrázek 2.1: Ukázky tvarů částicových systémů
a) kruh; b) bod; c) obdélník;
d) prstenec; e) bod, částice jsou neusále generovány; f) rotující bod

2.2.2 Parametry emitorů

Každý emitor má rozsáhlou sadu vlastních parametrů. Mohou to být parametry, které přímo neovlivňují chováni jednotlivých částic, např. kolik vygenerovat částic za jednotku času, intervaly generování apod. Dále to jsou parametry, kterými jsou částice ovlivněny. Jsou to především počáteční hodnoty, jenž jsou předány a nastaveny každé částici ve chvíli, kdy je na tomto emitoru vygenerována. Často nemají charakter přesných hodnot, ale jsou vyjádřeny průměrnou hodnotou a variaci, s jakou se může hodnota od průměru lišit (např. velikost 30 jednotek ± 20%).

2.3 Částice

Částice jsou základní a nejmenší prvky celého částicového systému. Jsou to na sobě nezávislé hmotné body v prostoru s řadou vlastností, jako jsou velikost, rychlost, pozice, barva, průhlednost, stáří, doba existence apod. Počáteční stav těchto vlastností bývá určen jako náhodná hodnota z určitého rozsahu. Tímto se zajistí, že každá částice má různé chování a simulace se tak přiblíží realitě.

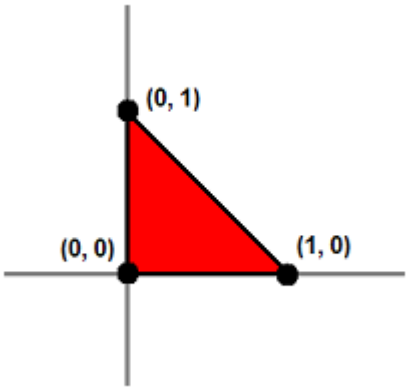
Některé z těchto vlastností mohou být v průběhu simulace ovlivňovány vnějšími vlivy. Tyto vlivy se dělí na globální a lokální. Globální nejsou závislé na poloze částice, což odpovídá homogenní gravitaci nebo větru. Naopak lokální vlivy na této poloze závisle jsou, například radiální gravitace (magnet, planeta ve vesmíru…), turbulence, vír apod.

Částice jsou vykreslovány jako některý ze základních geometrických útvarů (bod, koule, krychle, mnohostěn…). Velmi časté je použití tzv. *billboardingu* (viz kapitola 2.7).

2.4 Vertex

Jedná se v podstatě o bod v prostoru. Je nejzákladnějším primitivem, protože všechny ostatní primitiva, jako například úsečka, trojúhelník či polygon se skládají minimálně ze dvou vertexů.

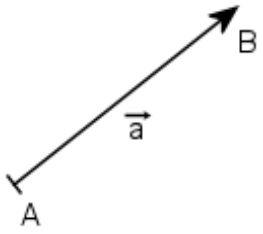
Vertex je definován svou polohou v prostoru (souřadnicemi x, y, z). Vzhledem k jeho bezrozměrnosti je zbytečné zaznamenávat další informace jako je rotace podle základních os nebo faktor zvětšení (*scale*). Může však obsahovat velké množství dalších informací důležitých k popisu objektu. Mezi ty nejdůležitější patří normálový vektor pro osvětlení, texturovací koordináty pro mapování textur nebo barevné složky.



Obrázek 2.2: Trojúhelník (face), tvořen třemi vertexy se souřadnicemi ve 2D prostoru

2.5 Vektor

Vektor je množina všech orientovaných úseček o stejné velikosti. Jedná se v podstatě o veličinu, která má kromě velikosti také směr, tímto se liší od skalárních veličin, které mají pouze hodnotu. Vektor bývá tištěn tučně (např. vektor $\boldsymbol{a}$ - takto jsou vektory značeny také v této práci). Při ručním psaní se často označuje $\vec{a}$ nebo $\underline{a}$. Graficky se vektor znázorňuje jako úsečka s šipkou na konci, jak je znázorněno na obrázku 2.3. Délka úsečky představuje velikost vektoru a směr úsečky představuje směr vektoru, viz [5].



Obrázek 2.3: Grafická reprezentace vektoru

2.5.1 Složky vektoru

Vektory jsou často vyjadřovány pomocí svých složek. To znamená, že vektor může být zapsán jako $\boldsymbol{a} = (a_1, a_2 \dots a_n)$, kde $a_1, a_2 \dots a_n$ jsou složky vektoru. Ty lze vypočítat takto:

$$\begin{aligned} a_1 &= B_1 - A_1 \\ a_2 &= B_2 - A_2 \\ &\vdots \\ a_n &= B_n - A_n \end{aligned} \tag{2.3}$$

Kde A je počáteční bod vektoru a B je koncový bod vektoru. $A_1, A_2 \dots A_n$ a $B_1, B_2 \dots B_n$ jsou souřadnice těchto bodů.

2.5.2 Velikost vektoru

Velikost vektoru lze spočítat podle vzorce 2.4. Jestliže je tato velikost rovna jedné, můžeme takový vektor označit jako jednotkový. Pokud je rovna nule, pak se tento vektor nazývá nulový. Takovýto vektor nemá z fyzikálního hlediska směr ani orientaci.

$$|\boldsymbol{a}| = \sqrt{a_1^2 + a_2^2 + \cdots + a_n^2} \tag{2.4}$$

Kde $a_1, a_2 \dots a_n$ jsou složky vektoru, jehož velikost chceme vypočítat.

2.5.3 Skalární součin

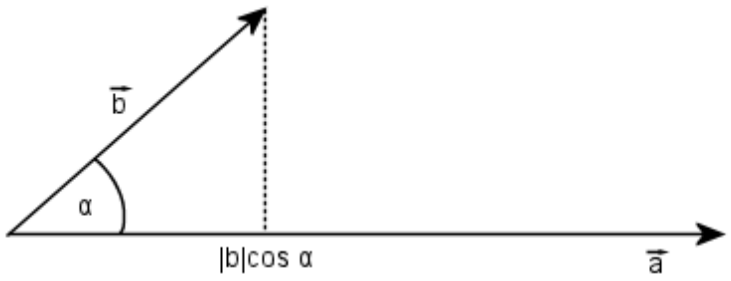
Skalární součin dvou vektorů $\boldsymbol{a}$ a $\boldsymbol{b}$ se obvykle značí jako $\boldsymbol{a} \cdot \boldsymbol{b}$. Výpočet lze provést pomocí vzorce 2.5 a jeho výsledkem je, jak název napovídá, skalár, tedy číselná hodnota. [6]

$$\boldsymbol{a} \cdot \boldsymbol{b} = a_1 b_1 + a_2 b_2 + \cdots + a_n b_n \tag{2.5}$$

Kde $a_1, a_2 \dots a_n$ a $b_1, b_2 \dots b_n$ jsou složky vektorů, jejichž skalární součin chceme spočítat.

Z geometrického hlediska představuje skalární součin vektorů $\boldsymbol{a}$ a $\boldsymbol{b}$ součin velikosti vektoru $\boldsymbol{a}$ a velikosti průmětu vektoru $\boldsymbol{b}$ do směru vektoru $\boldsymbol{a}$, tzn.:

$$\boldsymbol{a} \cdot \boldsymbol{b} = |\boldsymbol{a}| \cdot |\boldsymbol{b}| \cdot \cos\alpha \tag{2.6}$$



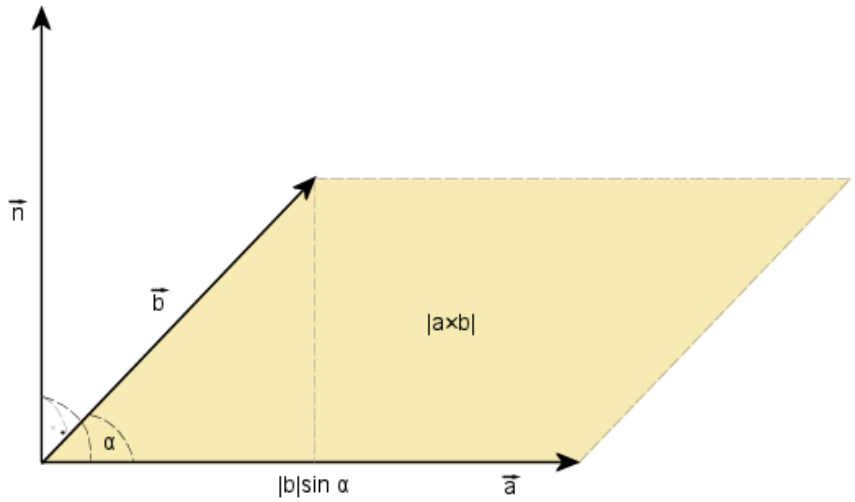
Obrázek 2.4: Geometrický význam skalárního součinu

2.5.4 Vektorový součin

Vektorový součin dvou vektorů $\boldsymbol{a}$ a $\boldsymbol{b}$ se značí $\boldsymbol{a} \times \boldsymbol{b}$. Výpočet pro trojrozměrný prostor lze provést podle vzorce 2.7. Na rozdíl od skalárního součinu je výsledkem vektorového součinu vektor a lze jej provést pouze v prostoru s více než dvěma rozměry. Výsledný vektor je kolmý k původním dvěma

vektorům a jeho velikost odpovídá obsahu plochy rovnoběžníka, který tyto vektory tvořily (viz obrázek 2.5).

$$\boldsymbol{n} = \boldsymbol{a} \times \boldsymbol{b} = (a_2 b_3 - a_3 b_2;\ a_3 b_1 - a_1 b_3;\ a_1 b_2 - a_2 b_1) \tag{2.7}$$



Obrázek 2.5: Vektorový součin

2.5.5 Normalizace

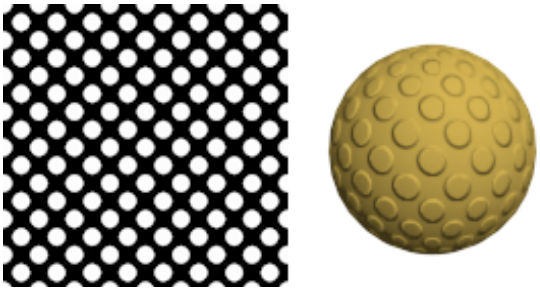
Při této operaci jsou složky vektoru změněny tak, aby měl jednotkovou velikost, ale zároveň aby se nezměnil jeho směr. Této operace se v počítačové grafice hojně využívá v případech, kdy chceme, aby vektor při násobení ovlivnil pouze směr výsledného vektoru, ale nikoli jeho velikost. Normalizace se dosáhne tak, že se každá složka vektoru podělí jeho délkou:

$$\boldsymbol{b} = \frac{\boldsymbol{a}}{|\boldsymbol{a}|} \tag{2.8}$$

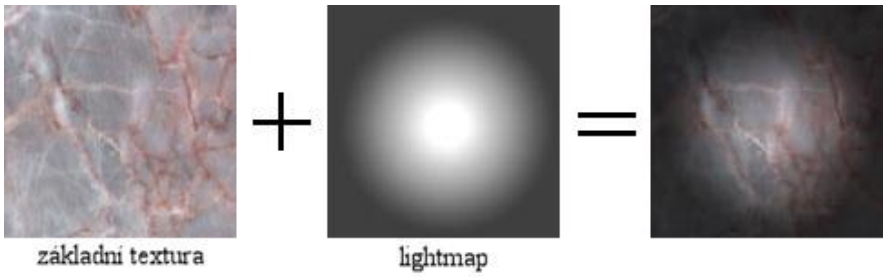
Výsledný vektor $\boldsymbol{b}$ je normalizovaný vektor $\boldsymbol{a}$. Je to jednotkový vektor se stejným směrem jako měl vektor $\boldsymbol{a}$.

2.6 Textura

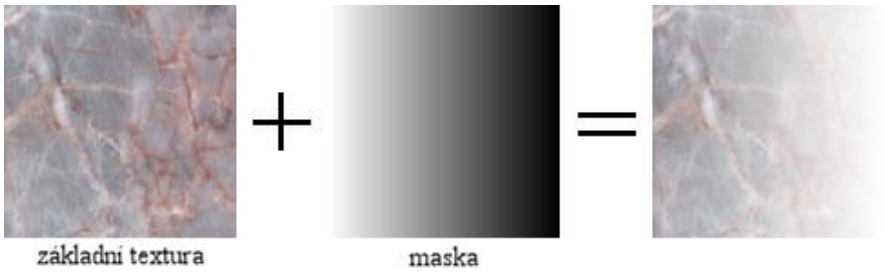
Textury jsou reprezentací povrchu dvourozměrného nebo trojrozměrného objektu. Konkrétně se jedná o kolekci *pixelů* (u textur označovaných jako *texel*), kde každý texel uchovává informaci o barvě. Tato barva se však vždy nemusí interpretovat jen jako barva povrchu, ale třeba jako jeho výška u *bumpmappingu* (viz obrázek 2.6), jas u *lightmappingu* (viz obrázek 2.7) nebo průhlednost (viz obrázek 2.8).



Obrázek 2.6: Bumpmapping



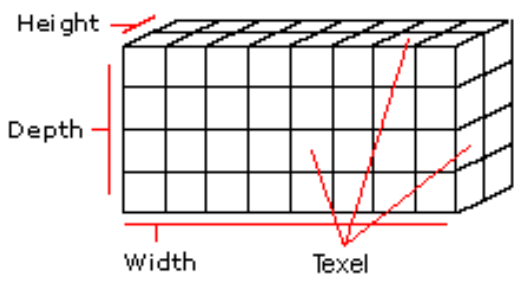
Obrázek 2.7: Lightmapping



Obrázek 2.8: Průhlednost

2.6.1 3D textura (volume textura)

Kolekce, která obsahuje jednotlivé texely, nemusí být vždy jen dvourozměrná. Pokud přidáme jeden rozměr, stává se ze standardní textury *3D textura*, která je někdy nazývána také *volume textura* (viz obrázek 2.9). Vertexy objektu, na který je tato textura mapována, musí však mít tři složky texturovacích koordinátů oproti obvyklým dvěma.



Obrázek 2.9: 3D textura (volume textura) o velikosti 8×2×4, obrázek převzat z[7]

2.6.2 Mapování textur

Jakmile je textura definována, je možné ji "obalit" okolo jakéhokoli trojrozměrného objektu. Tato metoda se nazývá *texturování* nebo také *mapování textury*. K mapování slouží texturovací koordináty, které jsou uloženy v každém vertexu trojrozměrnému objektu.

Pro různé druhy problémů se využívají různé způsoby mapování např. pro mapování na kouli (planety apod.) je vhodné použít tzv. *cubemapping*, pro mapování terénu vytvořeného z výškové mapy, explozí nebo mlhy se často využívají tzv. *volume textury* (viz kapitola 2.6.1).

2.6.3 Filtrování textur

Při vykreslování grafického prvku je mapován trojrozměrný objekt na dvourozměrnou obrazovku. Pokud má tento prvek texturu, musíme ji použít k určení barvy každého pixelu dvourozměrného obrazu tohoto prvku na obrazovce. Tento proces se nazývá *filtrování textur*.

Filtrování je více druhů. Paří mezi ně metoda "nearest-point", lineární, bilineární, anisotropické a filtrování pomocí mipmap. Každý z těchto typů má své výhody a nevýhody. Například lineární filtrování může způsobit zubaté okraje (*jagged edges*), má však nejnižší výpočetní náročnost. Oproti tomu použití mipmap většinou produkuje nejlepší výstup, zvláště v kombinaci s anisotropickým filtrování, ale také vyžaduje nejvíce výpočetních a paměťových zdrojů, viz [8].

2.7 Billboarding

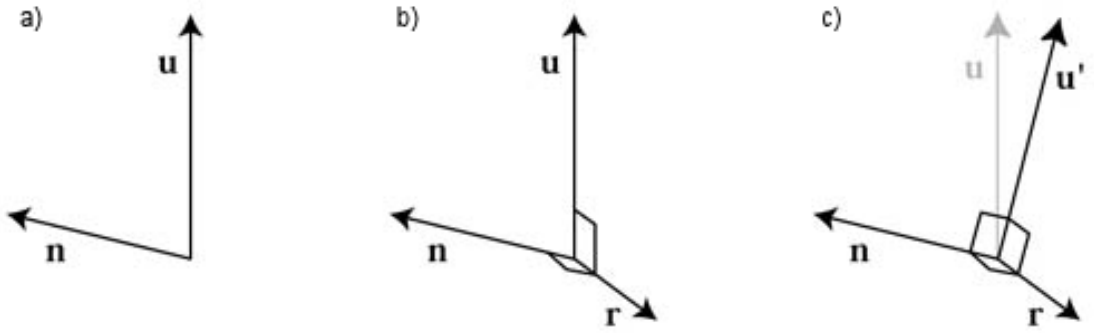
Jedná se o metodu, při které je orientace a tedy normálový vektor polygonu měněn v závislosti na směru pohledu. Většinou je měněn tak, aby směřoval vždy ke kameře a tudíž plocha polygonu byla vždy viditelná (*billboard*). Billboarding, kombinovaný s průhlednými texturami (*alpha texturing* případně *alpha blending*) a animací, je velmi často používán pro reprezentaci mnoha jevů, které nemají pevný povrch, například oheň, mlha, exploze nebo mraky.

K vytvoření rotační matice tak, aby po její aplikaci vznikl billboard, jsou zapotřebí tři navzájem kolmé vektory. Normálový vektor polygonu (billboardu), vektor, který ukazuje k pravému okraji polygonu (dále *right-vector*), a vektor, jenž byl určen programátorem a stanovuje, kde je v trojrozměrném světě "nahoru" (tzv. *up-vector*).

Normálový vektor povrchu a up-vector není obtížné získat. Normálový vektor povrchu je průměr normálových vektorů vertexů, které tento povrch definují a up-vector lze získat přímo z kamery (*view matrix*) nebo matice světa (*world matrix*). Výběr závisí na tom, zda chceme billboard zarovnaný s obrazovkou (*screen-aligned billboard*) nebo billboard zarovnaný se světem (*world-oriented billboard*). Ovšem získat vektor, který ukazuje k pravému okraji polygonu, není už tak snadné a je zapotřebí výpočet.

Ve všech billboardovacích technikách je jeden z těchto vektorů určen jako fixní a cílem je ostatní vektory upravit tak, aby na něj byly kolmé. Pokud si jako fixní vektor zvolíme normálový vektor povrchu, což je obvyklá praxe, je postup následující:

1. Prvně je vytvořen right-vector **r** jako vektorový součin normálového vektoru povrchu **n** a up-vectoru **u** (viz obrázek 2.10b).
2. Normalizujeme nově vytvořený right-vector **r**.
3. Nyní vypočítáme vektorový součin right-vectoru **r** a normálového vektoru povrchu **n** (obecně fixního vektoru), čímž získáme upravený up-vector **u'** (viz obrázek 2.10c).
4. Normalizujeme nový up-vector **u'**.



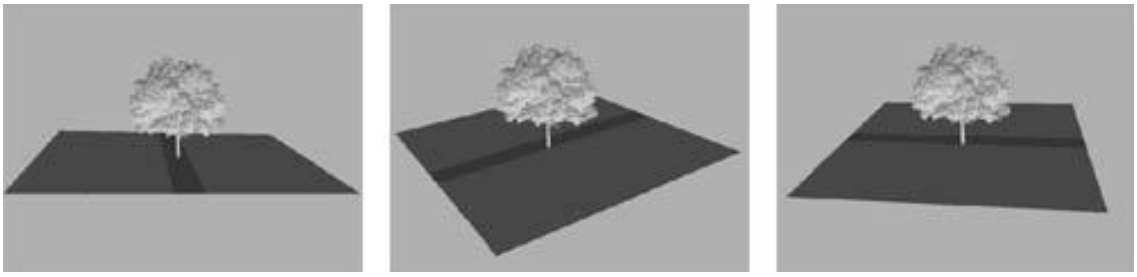
Obrázek 2.10: Výpočet vektorů pro billboarding

5. Z těchto vektorů vytvoříme rotační matici $\mathbf{T_r}$ (viz matice 2.9).

$$T_r = \begin{pmatrix} r_1 & r_2 & r_3 & 0 \\ u_1' & u_2' & u_3' & 0 \\ n_1 & n_2 & n_3 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{2.9}$$

6. Aplikujeme rotační matici.

V případě, že jako fixní vektor určíme up-vector, jedná se o takzvaný osový billboarding (*axial billboarding).* Tento typ se často využívá pro zobrazování stromů, paprsků a jiných prvků, jejichž rotace je fixována na jednu osu. Jak je vidět na obrázku 2.11, strom se neotáčí tak, aby normálový vektor jeho povrchu vždy směřoval do kamery, ale otáčí se okolo osy kmenu. Tedy okolo up-vectoru, který byl určen za fixní.



Obrázek 2.11: Zobrazení stromů pomocí axial billboardingu

Aby při billboardingu, který využívá alpha blending, nevznikaly nežádoucí defekty (tzv. artefakty), je potřeba jednotlivé billboardy seřadit sestupně podle vzdálenosti od kamery (od nejvzdálenějšího po nejbližší) a v tomto pořadí je vykreslovat. K tomuto řazení se zpravidla využívá některý z rychlých řadících algoritmů jako je *heapsort* a jeho varianty, viz [9].

2.8 Gravitace

Gravitace, nebo také gravitační interakce, je univerzální silové působení mezi všemi formami hmoty. Je nejslabší ze všech základních interakcí, má nekonečný dosah a je vždy přitažlivá. V současné době nejlépe popisuje gravitaci Obecná teorie relativity jako zakřivení časoprostoru. My si ovšem vystačíme s aproximací pomocí Newtonova gravitačního zákona. Ten říká, že každá dvě tělesa, která lze považovat za hmotné body nebo homogenní koule, na sebe působí gravitační silou F_g, kterou lze vyjádřit tímto vzorcem:

$$F_g = \varkappa \frac{m_1 m_2}{r^2} \tag{2.10}$$

Kde F_g je síla působící mezi dvěma hmotnými body, m_1, m_2 jsou hmotnosti hmotných bodů, r je vzdálenost mezi hmotnými body a $\varkappa$ je gravitační konstanta, která se rovná přibližně $\varkappa = 6{,}67 \cdot 10^{-11} N\, m^2\, kg^{-2}$.

2.8.1 Homogenní gravitační pole

Pojem homogenní gravitace vyjadřuje určité matematické zjednodušení popisu gravitačního pole. Při použití tohoto modelu má gravitační síla ve všech bodech stejnou velikost i směr. Toto zjednodušení lze aplikovat na případy, kdy se pole, v němž pozorovaný děj probíhá, příliš nemění. Například při pohybu těles blízko povrchu velkých vesmírných těles. Gravitační síla je pak vyjádřena následujícím vzorcem:

$$F_g = m \cdot g \tag{2.11}$$

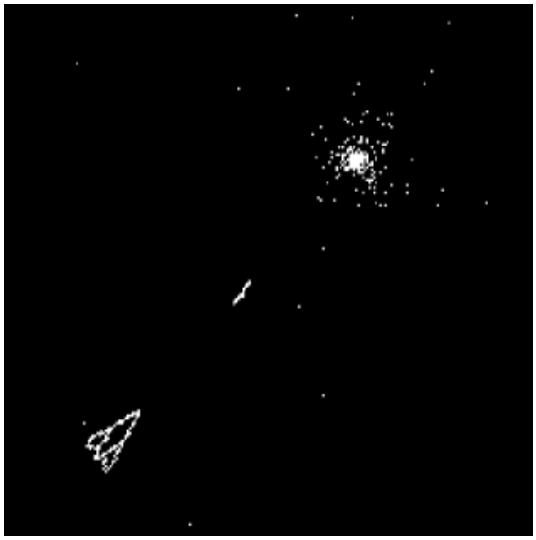
Kde F_g je gravitační síla působící na těleso, m je hmotnost tělesa a g je gravitační zrychlení (například u Země je $g = 9{,}83\ m\ s^{-2}$.)

2.8.2 Radiální (centrální) gravitační pole

Jedná se typ gravitačního pole, kde směr gravitační síly ve všech místech pole míří do jednoho bodu (středu). Tento typ lze výborně využít při aproximaci gravitačního pole vesmírných těles jako jsou planety a hvězdy.

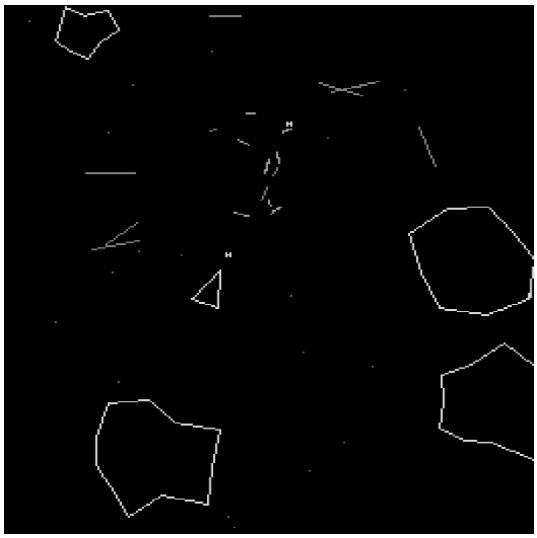
3 Historie

První velmi jednoduché částicové systémy se objevily na počátku 60. let, kdy světlo světa spatřily první počítačové hry. Jedna z těchto her se nazývala "Spacewar!" a využívala oblak pixelů k simulaci explozí (viz obrázek 3.1). Tyto "částice" měly náhodný směr i rychlost a nebyly ovlivněny žádnými vnějšími vlivy. [10]



Obrázek 3.1: Exploze ve hře Spacewar!

Pravděpodobně první "fyzikální" částicový systém se objevil v roce 1979, kdy Atari Inc. vydalo svou veleúspěšnou hru "Asteroids", která již dokonce využívala vektorovou grafiku. Právě pomocí této vektorové grafiky byly realizovány exploze (viz obrázek 3.2).



Obrázek 3.2: Exploze ve hře Asteroids

Termín "částicový systém" byl představen v roce 1982 Williamem T. Reevesem, kdy byl do kin uveden film "Star Trek II: The Wrath of Kahn". Ve filmu byla představena "zbraň" zvaná Genesis, schopná z nehostinné planety vytvořit planetu obyvatelnou (viz obrázky 3.3 a 3.4). Efekt této přeměny byl vytvořen právě pomocí částicového systému. Reeves tehdy definoval částicový systém takto (tato definice platí dodnes):

> *"A particle system is a collection of many many minute particles that together represent a fuzzy object. Over a period of time, particles are generated into a system, move and change from within the system, and die from the system."*
>
> *– Reeves, Particle Systems - A Technique for Modeling a Class of Fuzzy Objects*

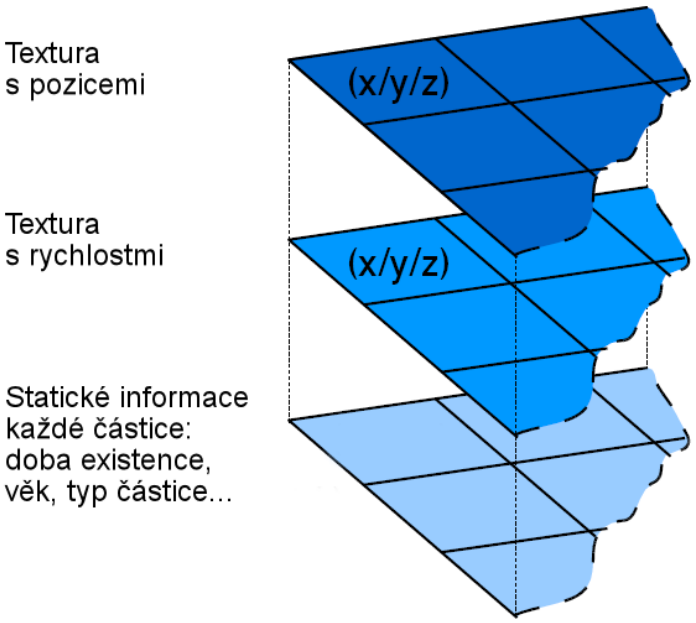
Obrázek 3.3: Genesis

Obrázek 3.4: Genesis

4 Současný trend

V současnosti se vývojáři aplikací (především se jedná o vývojáře počítačových her) simulujících chování částic snaží o co nejúčinnější hardwarovou akceleraci pomocí GPU (*Graphic Processing Unit*) grafických karet. Na rozdíl od CPU je GPU specializované pro práci s grafickými prvky a poskytuje možnost velmi rychlých paralelních výpočtu. Převedením simulace chování částic na GPU je dosaženo několikanásobného zrychlení a to nám umožňuje simulovat větší množství menších částic, což vede ke zvýšení realističnosti celé simulace. Další výhodou je snížení zátěže CPU, který se může věnovat jiným složitým výpočtům jako například fyzikálnímu modelu nebo umělé inteligenci.

Výpočty na GPU jsou prováděny pomocí *shaderů*. Existují tři druhy: pixel, vertex a geometry shader. K simulaci se nehodí vertex shadery, protože nedokážou uchovávat stav simulace a částic (pozici, rychlost atd.). Naproti tomu pokud reprezentujeme tyto stavy pomocí textur, můžeme s nimi pracovat s použitím pixel shaderů. Každá textura je úložiště pro jeden typ stavů částic a každý texel této textury je stav jedné částice. To znamená, že existuje jedna textura, která uchovává rychlost částic, druhá uchovává pozici a tak dále (viz obrázek 4.1).[10]



Obrázek 4.1: Textury reprezentující stavy částic

S těmito texturami je zacházeno jako s jednorozměrným polem. Je potřeba určit, který prvek v tomto poli je volný pro uložení nové (právě vygenerované) částice. Pro tento účel se obvykle používají rychlé alokátory, například zásobníky, do kterých se ukládají indexy volných prvků.

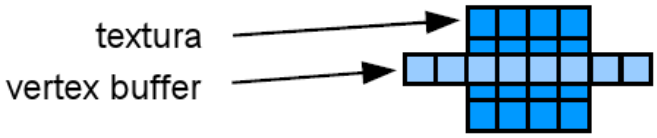
K aktualizaci stavů částice je zapotřebí použít jednu z jednoduchých integračních metod. Většinou je použita jedna z následujících tří: Eulerova metoda, Verletova metoda nebo Runge-Kutta

metoda vyššího řádu. Každá z těchto metod má své výhody a nevýhody. Eulerova metoda je výpočetně jednoduchá a nenáročná, ale potřebuje uchovávat pozici a rychlost částice. Verletova metoda potřebuje uchovávat jen pozici, ale dva kroky zpátky a manipulace s rychlostí (např. kvůli kolizím) je téměř nemožná.

Pixel shader je poté využit k provedení jednoho iteračního kroku. To znamená, že je pomocí něj vypočítána nová rychlost a poté pozice. Pokud hardware podporuje tzv. MRT (*Multiple Render Targets*), je možné provést oba kroky zároveň.

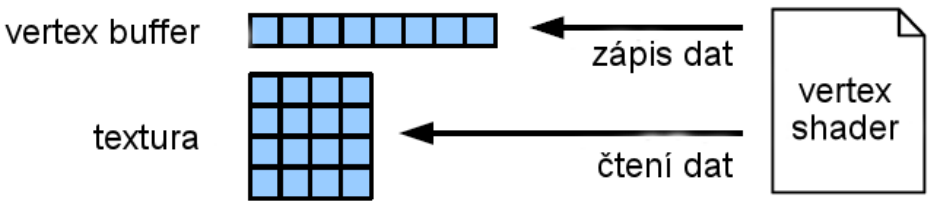
Abychom byli schopni vykreslit částice na jejich pozicích, ku příkladu pomocí billboardingu, musíme data uložená v texturách převést na vertexy a uložit je do vertex bufferu. Novější a výkonnější metoda překopíruje data z textury do vertex bufferu (viz obrázek 4.2) nebo dokonce přímo interpretuje data textury jako data vertexů (viz obrázek 4.3). Tuto metodu však podporuje pouze nejnovější hardware v kombinaci s balíkem knihoven DirectX 10.0 od společnosti Microsoft, který lze použít pouze na operačním systému Windows Vista a vyšších. Podporuje ji i OpenGL skrze určitá rozšíření.

Obrázek 4.2: Kopírování dat z textury přímo do vertex bufferu



Obrázek 4.3: Interpretování dat textury jako data vertex bufferu

Druhá metoda je méně výkonná, ovšem lze ji implementovat i pomocí nižší verze DirectX a OpenGL. Spočívá v přístupu k textuře z vertex shaderu a aktivním čtení jejích dat, ze kterých poté shader vytvoří vertex buffer (viz obrázek 4.3).



Obrázek 4.4: Převod dat z textury na vertex buffer

5 Návrh řešení

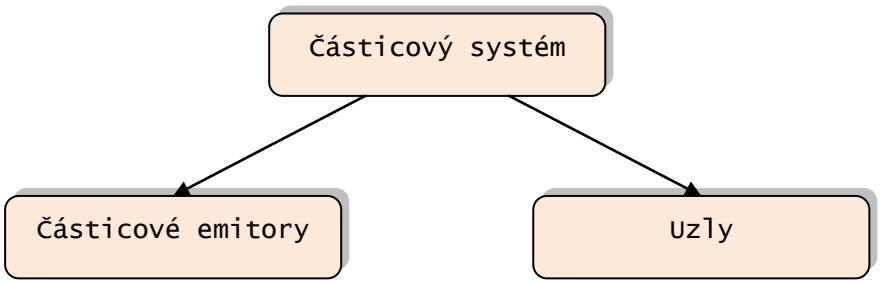
Při návrhu řešení jsem se soustředil především na co nejmenší zatížení procesoru, i když to někdy mohlo vést ke zvýšení paměťových nároků. Ovšem v dnešní době, kdy třeba i kancelářský počítač má předimenzované množství operační paměti, je více nedostatkový zvláště výpočetní čas. Právě proto některé mé struktury a třídy přímo nedodržují zásady objektově orientovaného modelu.

V této kapitole bude vysvětleno rozdělení projektu na dva celky, způsob generování částic tak, aby nebyl závislý na FPS (*Frames Per Second*), organizace částic a způsob uchování jejich stavu, gravitace, detekce kolize a reakce na ni a další.

5.1 Členění projektu

Celý projekt jsem rozdělil na dva na sobě nezávislé celky. Jeden celek se stará o simulaci (tzn. generování částic, jejich dobu existence v systému, pohyb, rychlost, rotaci, směr, lokální a globální vlivy, detekce kolize apod.) a druhý celek těmto nasimulovaným částicím přiřadí pozici ve scéně, tvar, texturu, nastaví jejich průhlednost, v případě billboardingu jim vypočítá správnou rotační matici, seřadí je podle vzdálenosti od kamery a především je vykreslí na obrazovku.

Zástupci prvního celku jsou různé typy emitorů, rozlišené podle počátečních podmínek pro generování částic a jejich dalšího chování. Druhý celek pak reprezentují uzly (tzv. *nodes*), které se dělí podle geometrického útvaru (billboard, koule, krychle…), jenž je vykreslován na pozici částice. Instance zástupců obou těchto celků jsou vytvářeny pomocí hlavní třídy částicového systému.



Obrázek 5.1: Návrh rozdělení projektu

Toto rozdělení umožňuje projekt snadno rozšiřovat o nové typy částicových emitorů a uzlů. Je také možné zahodit všechny původně vytvořené a naprogramovat si své vlastní.

5.2 Simulace

V této kapitole vysvětlím návrhy některých struktur a tříd, které jsem použil při simulování chování částic. Princip generování částic, určení náhodného bodu v geometrickém útvaru, výpočet rozptylu částic, přiřazení gravitace a odrazových ploch (deflektorů) emitoru částic, reakce na kolizi s deflektorem a v neposlední řadě organizace částic v systému. Dále také objasním snahu o vytvoření kódu s co nejmenším počtem volání metod a funkcí.

5.2.1 Emitor částic

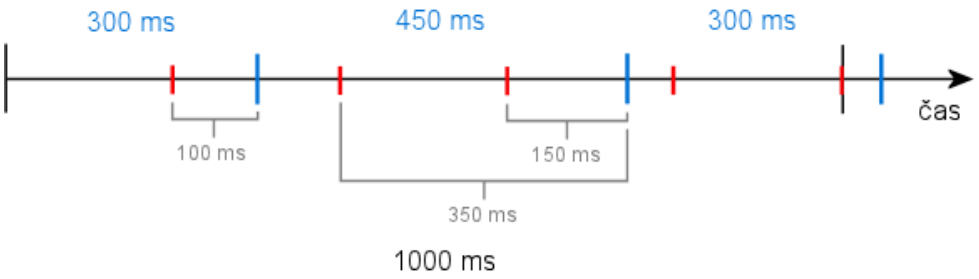
Jednotlivé typy emitorů částic jsou zděděny z jedné hlavní třídy, ve které je implementováno veškeré chování částic po vygenerování. Liší se konstruktory, které přijímají jiné parametry, a jedinou metodou. Konstruktory přijímají jako parametry základní atributy emitoru, jako například poloměr kruhu, strana čtverce nebo rozptyl. Metoda nastavuje počáteční vlastnosti všem částicím, které se na tomto emitoru vygenerují. U většiny emitorů je nastavena nová pozice a vektor rychlosti. Například u emitoru, kde se částice generují na ploše elipsy, je nutné určit náhodný bod uvnitř této elipsy a upravit směr vektoru rychlosti, podle zadaného rozptylu.

5.2.1.1 Generování částic

Velmi důležitým prvkem celé simulace je plynule generovat částice. Toto generování nesmí být závislé na počtu rámců za vteřinu (FPS) a v době generování musí být částice umístěny na správnou pozici a nastaven správný věk, neboť perioda vykreslování se vždy nemusí shodovat s periodou generování. Při malém FPS se může dokonce stát, že v době generování již částice vůbec neexistuje. V tom případě je nutné zajistit, aby se částice zbytečně nevytvářela, protože bychom ji v příštím průchodu stejně odstraňovali. Postup generování částic ukážu na následujícím příkladě.

Obrázek 5.2: Časová osa s okamžiky generování částic

Mějme emitor částic, který má generovat 5 částic za sekundu a délka existence částice je 300 milisekund. Perioda generování (doba, za kterou je vygenerována jedna částice) je tedy 200 milisekund. Na obrázku 5.2 jsou na časové ose **červeně** vyznačeny okamžiky, kdy je potřeba vygenerovat částici.



Obrázek 5.3: Časová osa s okamžiky vykreslování

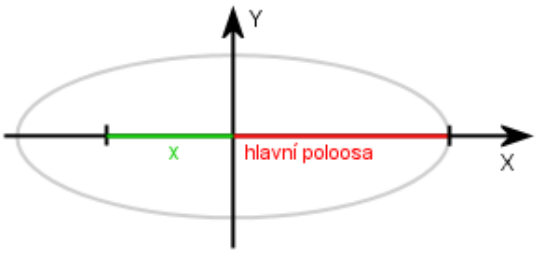
Na obrázku 5.3 jsou **modře** vyznačeny momenty, kdy dochází k vykreslování scény a je nutné vygenerovat korektní počet částic a nastavit jim věk podle toho, jak dlouho by již měly existovat. Například první částice by se měla objevit 200 milisekund po aktivaci emitoru. Avšak k vykreslení dojde až 300 milisekund po aktivaci, z čehož plyne, že částici musíme nastavit tak, že existuje již 100 milisekund a podle toho také aktualizovat její vlastnosti proměnné v čase (pozice, rotace, barva...).

U druhého vykreslení scény je zapotřebí vytvořit dvě částice. První částice je však v době vykreslování již "mrtvá", neboť by v tu dobu existovala 350 milisekund, což je více než je maximální doba existence nastavená v emitoru. Druhá částice je vygenerována a její věk je nastaven na 150 milisekund. U dalších vykreslování se postupuje obdobně.

5.2.1.2 Určení náhodného bodu

Částice jsou generovány v oblastech reprezentovaných určitými geometrickými útvary. Mezi dvourozměrné patří čtverce, kruhy a elipsy. Trojrozměrné jsou například krychle, koule atd. Proto je nutné určit náhodný bod v prostoru tohoto útvaru. Následující postup ukazuje, jak náhodně určit bod uvnitř elipsy. Všechny ostatní tvary jsou řešeny obdobně.

1. Vygenerujeme náhodné číslo z intervalu $\langle -hlavníPoloosa, +hlavníPoloosa \rangle$, které si označíme jako x. Toto číslo je poloha částice na ose X.

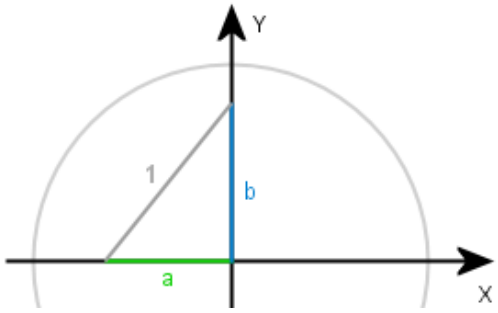


Obrázek 5.4: Elipsa, hlavní poloosa, náhodé číslo a

2. Číslo x podělíme velikostí hlavní poloosy. Tím získáme číslo v rozsahu -1 až 1, které označíme jako a.
3. Nyní využijeme Pythagorovu větu, kde velikost přepony určíme jako 1 a vypočítáme koeficient pro vedlejší poloosu, který nazveme b.

$$b = \sqrt{1 - a^2} \tag{5.1}$$

Výsledkem je číslo z intervalu $\langle 0, 1 \rangle$. Na jednotkové kružnici si tuto situaci lze představit následovně.

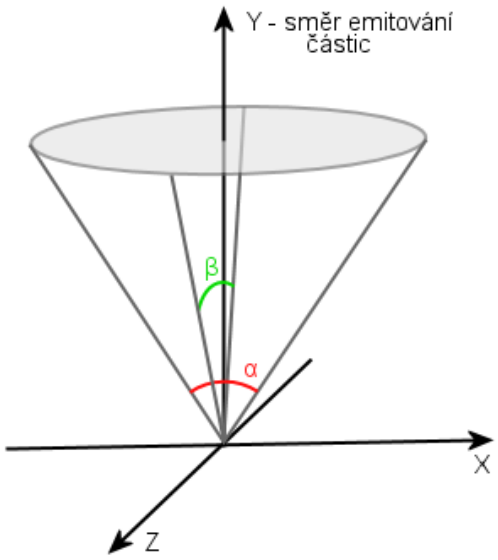


Obrázek 5.5: Jednotková kružnice

4. Vygenerujeme náhodné číslo z intervalu $\langle -vedlejšíPoloosa, +vedlejšíPoloosa \rangle$ a vynásobíme jej koeficientem b. Výsledné číslo je poloha částice na ose Y, proto jej označíme jako y.
5. Nastavíme novou polohu částice jako součet polohy emitoru a nově vypočítaného bodu o souřadnicích $[x, y]$.
6. Tuto polohu je ještě nutné transformovat rotační maticí emitoru, aby byla plocha, na kterou částici generujeme správně orientovaná.

5.2.1.3 Výpočet rozptylu částic

Předpokládáme, že máme dva úhly α a β, které specifikují rozptyl, tak jak je vidět na obrázku 5.6.



Obrázek 5.6: Rozptyl částic

1. Vygenerujeme dva náhodné úhly. První z intervalu $\langle -\frac{\alpha}{2}, \frac{\alpha}{2} \rangle$ a druhý z $\langle -\frac{\beta}{2}, \frac{\beta}{2} \rangle$ a označíme je postupně α_r a β_r.
2. Vytvoříme pomocný jednotkový vektor, jehož směr je směr rychlosti, kterou by částice měla, kdyby byly úhly α a β nulové.
3. Tento vektor otočíme o úhel α_r kolem osy Z a o úhel β_r okolo osy X. Dále jej transformujeme rotační maticí emitoru.
4. Nakonec výsledný vektor vynásobíme velikostí rychlosti částice a nastavíme jej jako vektor rychlosti částici.

5.2.2 Částice

Struktura, která vyjadřuje částici je typickým příkladem, kde byla obětována operační paměť ve prospěch rychlejšího přístupu, omezení časově náročných matematických operací a celkového ušetření výpočetního času CPU. Každá částice si uchovává tyto informace:

- pozice
- rotace
- rychlost
- barva
- velikost
- doba "života" (existence)
- věk
- násobič gravitace
- změna velikosti za milisekundu
- změna rotace za milisekundu pro každou osu zvlášť
- index další částice
- index předchozí částice

Změnu velikosti (rotace) spočítáme jen jednou při vygenerování částice jako:

$$\frac{koncováVelikost(koncováRotace) - počátečníVelikost(počátečníRotace)}{dobaExistence} \tag{5.2}$$

Tímto krokem spotřebujeme sice větší množství operační paměti, protože musíme pro každou částici ukládat data navíc, nicméně ušetříme výpočetní čas, neboť nemusíme pokaždé tuto změnu znovu počítat.

5.2.3 Gravitace

Každá gravitace je samostatný objekt, který je vytvářen nezávisle na emitoru částic. Uchovává informace o typu, síle a směru resp. středu gravitace.

Každý emitor částic má seznam odkazů na objekty gravitace, které na něj působí. Tímto je zajištěno, že na každý emitor může působit jiný počet gravitací, ale také, což je v praxi častější, jedna gravitace může působit na více emitorů. Při aktualizaci částice je každá gravitace v seznamu aplikována a částici je tak upravena velikost a směr rychlosti.

Síla gravitace přitom nezáleží jen na hodnotě, která je uložena v objektu gravitace samotném, ale také na násobiči gravitace, který má každá částice svůj. Takto lze například simulovat dým za ohnivou koulí. Zatímco je ohnivá koule plně ovlivněna gravitací, neboť je její násobič roven jedné, dým zůstává tam, kde se objevil nebo stoupá nahoru, jelikož je jeho násobič menší nebo roven nule.

Gravitace je reprezentována třídou, která má své atributy veřejné. Klasický přístup, kde má třída atributy soukromé a přistupuje se k nim přes veřejné metody, by byl v tomto případě, vzhledem k velmi častému volání metod, výpočetně náročný a tudíž nevhodný.

5.2.4 Odrazové plochy (deflektory)

Podobně jako gravitace jsou deflektory reprezentovány samostatným objektem, který je vytvářen nezávisle na emitoru částic. Uchovává normálový vektor roviny a bod na této rovině. Těmito dvěma prvky lze podle vzorce 4.2 určit rovinu v prostoru. Dalšími doplňujícími vlastnostmi jsou faktor změny odrazové rychlosti a procentuální vyjádření počtu částic, které se mají od deflektoru odrazit.

$$n_1 a_2 + n_2 a_2 + n_3 a_3 + d = 0 \tag{5.3}$$

Kde n_1, n_2, n_3 jsou složky normálového vektoru rovina, a_1, a_2, a_3 jsou souřadnice bodu A ležícího na rovině a d je vzdálenost bodu A od počátku souřadnic.

Stejně jako u gravitace má každý emitor částic speciální seznam, ve kterém uchovává odkazy na deflektory. Tento seznam je při aktualizaci každé částice prohlédnut a podle následujícího algoritmu je zjištěno, zda byl některý z deflektorů protnut.

1. Vypočítáme novou pozici částice.
2. Ze seznamu vybereme další deflektor, který by mohla částice protnout. Pokud je seznam prázdný, pokračujeme bodem 6.
3. Zjistíme vzájemnou polohu částice a aktuálního deflektoru na staré i na nové pozici pomocí následujícího vzorce

$$\frac{n_1 a_1 + n_2 a_2 + n_3 a_3 + d}{\sqrt{n_1^2 + n_2^2 + n_3^2}} = \frac{\boldsymbol{n} \cdot \boldsymbol{a} + d}{\sqrt{n_1^2 + n_2^2 + n_3^2}} \tag{5.4}$$

kde n_1, n_2, n_3 jsou složky normálového vektoru roviny $\boldsymbol{n}$, a_1, a_2, a_3 jsou složky vektoru $\boldsymbol{a}$, který určuje pozici částice, d je vzdálenost deflektoru od počátku souřadnic.

4. Pokud jsou znaménka výsledků stejná, nedošlo ke kolizi a pokračujeme bodem 2, jinak pokračujeme bodem 5.
5. Jelikož částice může protnout více deflektorů, uložíme si tento aktuální deflektor do pomocné proměnné. To však pouze v případě, že je blíže částici než deflektor, který je uložen v této proměnné nyní. Pokračujeme bodem 2.
6. Pokud nebyl protnut žádný deflektor, ukončíme algoritmus.
7. Nyní víme, že byl deflektor protnut, dokonce víme který (uložen v pomocné proměnné), ale nevíme kdy přesně. Tento čas přibližně určíme pomocí půlení intervalu.

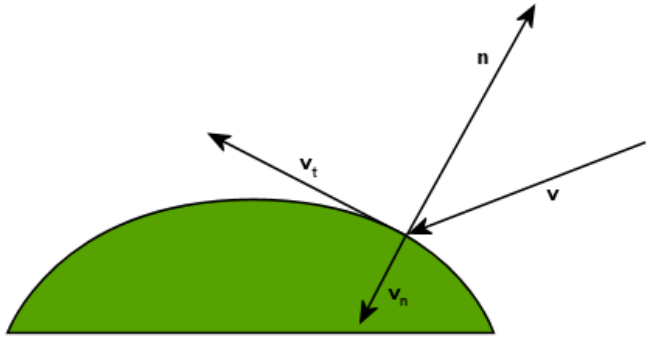
Po detekování kolize je nutné na tuto kolizi nějak reagovat. Obvyklé reakce jsou odraz, vyřazení částice ze systému nebo "přilepení" částice na deflektor. Následující postup ukazuje, jak určit novou velikost a směr rychlosti po odrazu. [10]

1. Rozložíme rychlost částice na normálovou a tečnovou komponentu (vzhledem k deflektoru) pomocí vzorců 5.5 a 5.6, jak ukazuje obrázek 5.7.

$$\boldsymbol{v_n} = (\overline{\boldsymbol{v}} \cdot \boldsymbol{n})\boldsymbol{n} \tag{5.5}$$

$$\boldsymbol{v_t} = \overline{\boldsymbol{v}} - \boldsymbol{v_n} \tag{5.6}$$

kde $\boldsymbol{v_n}$ je normálová složka rychlosti, $\overline{\boldsymbol{v}}$ rychlost částice, $\boldsymbol{n}$ je normálový vektor deflektoru v bodě dopadu částice, $\boldsymbol{v_t}$ je tečnová složka rychlosti.

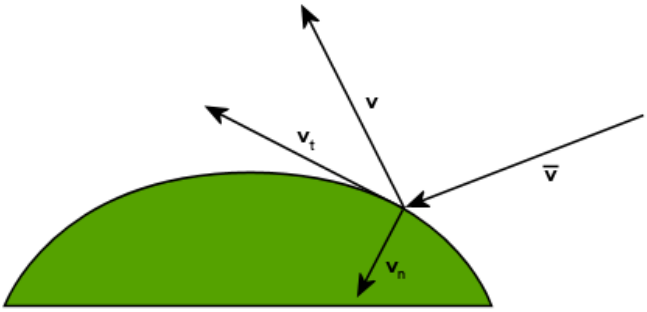


Obrázek 5.7: Rozklad rychlosti částice

2. Výslednou rychlost poté určíme podle vztahu 5.7, detail na obrázku 5.8.

$$\boldsymbol{v} = \boldsymbol{v_t} - \epsilon \boldsymbol{v_n} \tag{5.7}$$

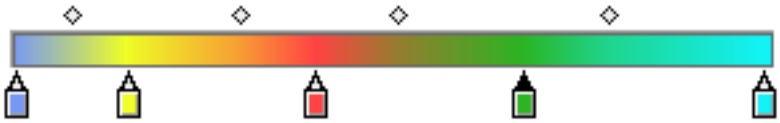
kde $\boldsymbol{v}$ je výsledná rychlost odražené částice, $\boldsymbol{v_t}$ je tečnová složka rychlosti, ϵ je faktor změny odrazové rychlosti, $\boldsymbol{v_n}$ je normálová složka rychlosti.



Obrázek 5.8: Výsledná rychlost po odrazu částice

5.2.5 Barevný přechod

Abychom dosáhli co nejlepších vizuálních efektů, nevystačíme si pouze s jednou barvou částice. Je potřeba, aby se barva v průběhu jejího života plynule měnila. K zaznamenání této změny slouží tzv. barevný přechod (*gradient*). Jedná se v podstatě o úsečku, na které jsou vyznačeny body, jež uchovávají barvu na daném místě. Mezi těmito body jsou barvy, které vzniknou lineární interpolací mezi barvami těchto bodů. Například takto vypadá vizualizace barevného přechodu v aplikaci Paint Shop Pro X2:



Obrázek 5.9: Barevný přechod v aplikaci Paint Shop Pro X2

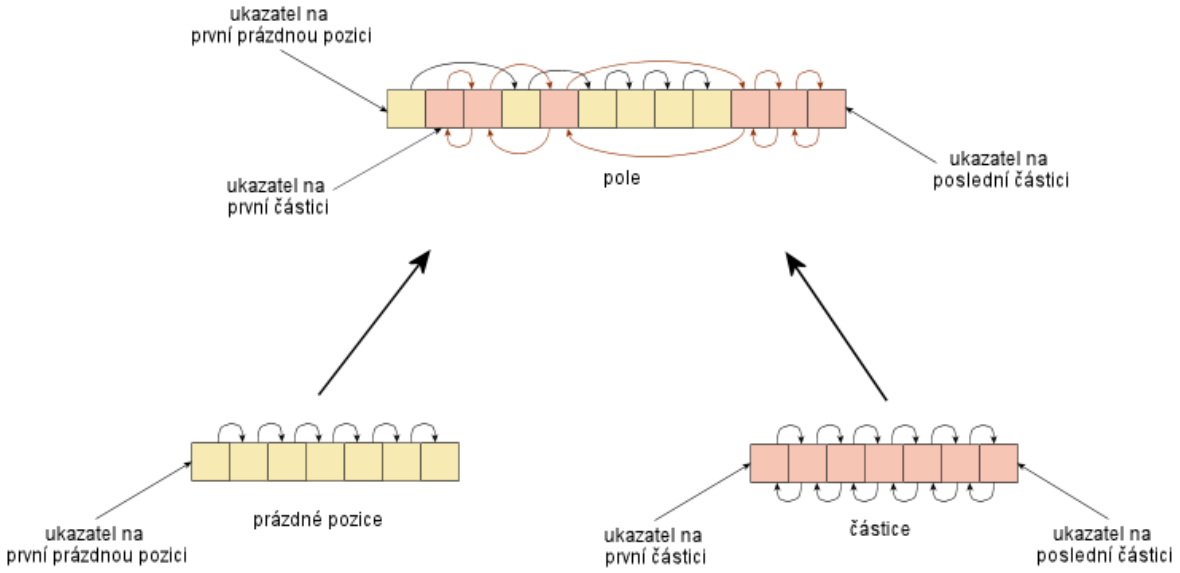
Při návrhu třídy, která by tento barevný přechod reprezentovala, jsem se opět soustředil na co největší zkrácení doby potřebné pro získání barvy z libovolného místa přechodu. Celý přechod jsem rozdělil na tisíciny a pro každou tuto část jsem vždy při přidání nebo odebrání bodu interpoloval její novou barvu. Toto může vypadat jako špatný přístup, který nám snížení výpočetních nároků nezajistí, ale tyto operace se provádějí před celou simulací a jen jednou. Při samotné simulaci již není nutné tuto časově náročnou interpolaci provádět a barvu částice v konkrétním čase můžeme získat velice snadno a rychle.

5.2.6 Organizace částic

Pro uchování částic v paměti bylo zvoleno statické pole, jehož velikost je spočítána jednou při vytvoření emitoru částic. Velikost tohoto pole je stanovena jako maximální možný počet částic, který může být v systému v jednom časovém okamžiku. Výpočet této hodnoty je určen vzorcem 4.2. To zaručuje, že jsou částice v paměti za sebou a není nutné používat ukazatele, jak by tomu bylo při použití například standardního C++ seznamu.

$$maximálníDobaExistenceČástice \cdot částicZaMilisekundu \tag{5.8}$$

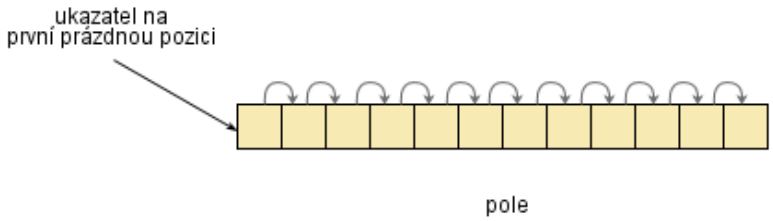
V tomto statickém poli existují dva seznamy. Jeden je seznam, který uchovává prázdné pozice, kam je možné uložit právě vygenerovanou částici. Uvolněná místa po zaniklých částicích se přidávají na začátek tohoto seznamu. Stejně tak zaplňování volných míst se provádí od začátku ke konci. Z toho jasně plyne, že na tyto úkony postačí jednosměrný seznam. Druhý seznam obsahuje částice, které jsou v současné chvíli v systému. Při vytváření jsou nové částice zařazovány na začátek seznamu, ovšem jelikož částice mohou zanikat v jiném pořadí, než byly vytvořeny, a tedy třeba uprostřed seznamu, je nutné, aby byl seznam obousměrný. Nejlépe to vysvětlí následující obrázek.



Obrázek 5.10: Organizace částic

5.2.6.1 Inicializace seznamu

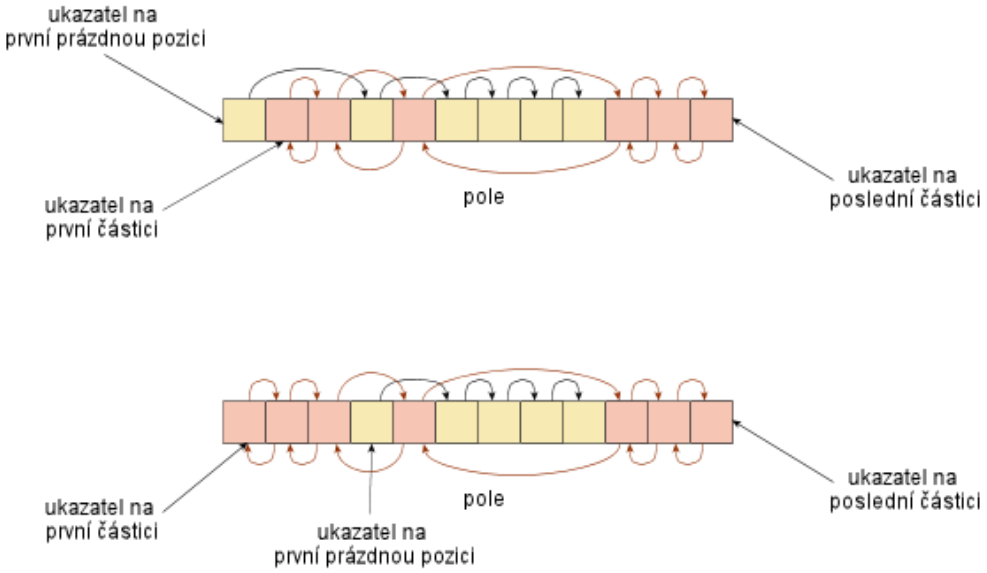
Při inicializaci je nastaven ukazatel na první prázdnou pozici na první prvek v poli (teda na 0). Ukazatelé na první a poslední částici jsou nastaveny tak, aby neukazovaly nikam. Všechny ostatní pozice jsou nastaveny jako prázdné a ukazují na další pozici. Po inicializaci vypadá pole jako na následujícím obrázku.



Obrázek 5.11: Inicializované pole

5.2.6.2 Přidávání částic

Nová částice je přidána na začátek seznamu částic a její data jsou uložena na první pozici v seznamu prázdných pozic. Ukazatel na první částici je nastaven na tuto novou částici a ukazatel na první volnou pozici je změněn na další volnou pozici.

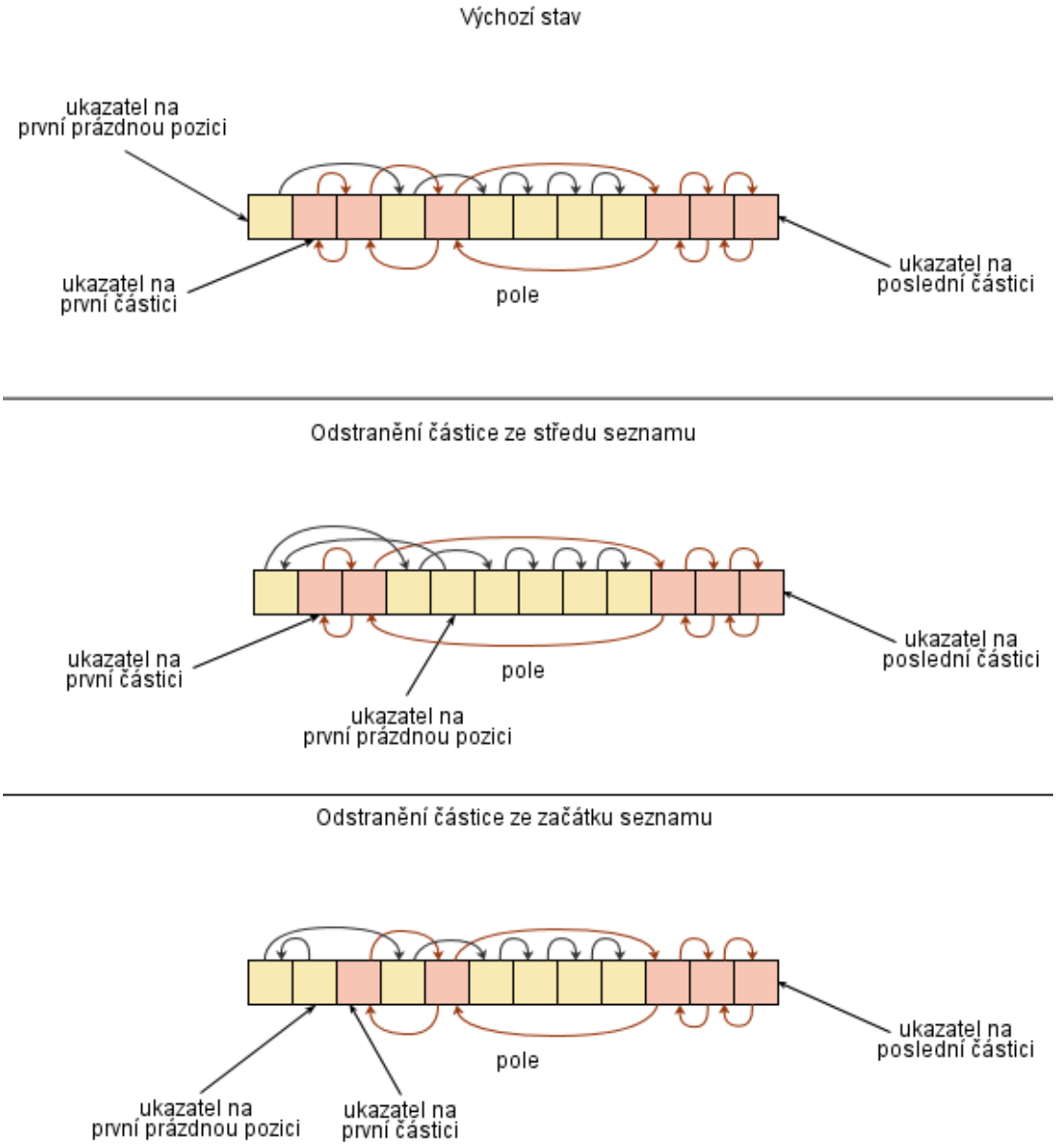


Obrázek 5.12: Přidání částice

5.2.6.3 Odebírání částic

Při odstraňování částice je nutné "překlenout" vzniklou mezeru v seznamu. To znamená, že ukazatel předchozí částice na další částici se nastaví na další částici za mazanou částicí a ukazatel následující částice na předchozí částici je nastaven na částici, která je před mazanou částicí. Nově vzniklé místo se přidá na začátek seznamu volných pozic. V případě, že je mazána částice na začátku nebo na konci

seznamu, je ukazatel na první resp. poslední částici změněn na další resp. předchozí částici od mazané. Situaci nejlépe osvětlí následující obrázky.



Obrázek 5.13: Odebírání částice

5.3 Vykreslování

Tato kapitola popisuje návrh řešení vykreslování jednotlivých částic. Návrh uzlů, jejich funkci v projektu a řazení podle vzdálenosti od kamery.

5.3.1 Uzly (nodes)

Jednotlivé typy uzlů jsou reprezentovány třídami, jež jsou děděny z jedné nadřazené třídy, obsahující atributy, které mají různé typy uzlů společné. To je především odkaz na emitor částic, jehož částice má uzel vykreslovat. Dále to je pozice a rotace ve scéně, pravdivostní hodnota zda použít barevný přechod apod. Tyto třídy se liší aktualizační a vykreslovací metodou.

V aktualizační metodě uzlu je zavolána aktualizační metoda emitoru, čímž je provedena simulace podle času, který uplynul od minulé aktualizace. Dále jsou v této metodě vytvořeny index a vertex buffery, vypočítány všechny vektory, matice a další prvky potřebné pro vykreslení primitiv reprezentujících částice.

5.3.2 Řazení podle vzdálenosti od kamery

Toto řazení lze využít především pro uzly, které nějakým způsobem využívají billboarding a alpha blending. Pro samotné řazení jsem využil max-heapsort, který je již součástí standardních knihoven jazyka C++. Například při simulaci dýmu není řazení potřeba, a proto lze nastavit pro každý uzel zvlášť.

6 Implementace

Projekt je implementován jako knihovna v jazyce C++ a tudíž nemá žádný vstupní bod. Pro jeho využití stačí v linkeru nastavit správnou cestu a do vašeho zdrojového kódu načíst hlavičkový soubor `mercury.h`.

Třídy projektu jsou rozděleny podle funkce do několika jmenných prostorů (*namespaces*):

- `mercury` hlavní jmenný prostor, který zapouzdřuje všechny ostatní jmenné prostory a také obsahuje hlavní třídu `cParticleEngine`
- `mercury::core` zapouzdřuje pomocné třídy a struktury, jako jsou `sVector3D`, `sColor`, `sMatrix4`, `sParticle` a další
- `mercury::math` obsahuje pomocné matematické funkce. Například `Random`, pro generování náhodných čísel, `Signum` pro určení znaménka atd.
- `mercury::emitters` obsahuje třídy všech typů emitorů částic. Například `cSprayPoint`, `cExplosionTrail`, `cParticleEmitter`
- `mercury::nodes` zapouzdřuje veškeré typy uzlů. Patří mezi ně `cBillboardNode`, `cParticleSystemNode`
- `mercury::gravity` obsahuje třídu reprezentující gravitaci
- `mercury::deflectors` tento jmenný prostor zapouzdřuje všechny typy deflektorů

Hlavní třídou projektu je `cParticleEngine`, která vytváří, spravuje a také uvolňuje z paměti všechny typy emitorů částic, gravitací, deflektorů a uzlů. Metody, které toto vytváření poskytují začínají slovem `Create`. Patří mezi ně `CreateSprayPoint`, `CreateBillboardNode`, `CreateGravitySpherical`, `CreateDeflectorPlane`, `CreateBillboardNode` a další. Tyto metody přijímají parametry, které definují nově vytvářený objekt. Například při vytváření radiální gravitace (`CreateGravitySpherical`) metoda přijímá souřadnice středu gravitace a sílu gravitace. Návratovou hodnotou těchto metod je odkaz na nově vytvořený objekt.

Třída si vytváří seznamy těchto objektů, aby mohla při volání metody `FreeAll` nebo při rušení uvolnit všechny objekty z paměti. Tato metoda se automaticky volá z destruktoru.

Všechny typy emitorů jsou zděděny ze třídy `cParticleEmitter`, která v konstruktoru přijímá parametry, které mají všechny emitory stejné. Například počet částic za sekundu, doba existence částice, rychlost, barva apod. Obsahuje také pole částic, které jsou emitorem generovány, a aktualizační metodu `Update`, ve které probíhá veškerá simulace chování částic.

Typy uzlů jsou také děděny z jedné nadřazené třídy, která se nazývá `cParticleSystemNode`. Konstruktor přijímá především odkaz na emitor částic, jehož částice bude vykreslovat a zařízení DirectX, které k tomuto vykreslení má použít. Jednotlivé typy se liší aktualizační metodou `Update` a vykreslovací `Render`.

Další třídy a struktury jsou většinou pomocné a k jejich vytvoření není potřeba volat žádnou metodu ve třídě `cParticleEngine`. Patří mezi ně struktury vektorů `sVector2D` a `sVector3D`, struktura pro čtvercovou matici 4×4 `sMatrix4`, struktury pro uchování barevných složek `sColor` a `sColorF` nebo třeba třída reprezentující barevný přechod `cColorGradient`.

Projekt je implementován pomocí rozhraní DirectX 9 od společnosti Microsoft. A pro jeho správnou kompilaci je zapotřebí mít nainstalovaný balík knihoven Microsoft DirectX SDK pro verzi 9.

7 Závěr

Cílem této bakalářské práce je seznámit čtenáře s některými základními pojmy týkajících se počítačové grafiky. Umožnit mu hlouběji proniknout do teorie částicových systémů, poznat jejich historii a současný trend. Na konci práce jsem předvedl návrh a implementaci jednoduchého částicového systému, který je navrhnut tak, aby co nejméně zatěžoval procesor, byť někdy na úkor paměťových nároků.

V první části práce je srozumitelně vysvětleno množství základních i méně základních pojmů, souvisejících s počítačovou grafikou, částicovými systémy, ale i matematikou a fyzikou. Tyto pojmy by měly čtenáři usnadnit pochopení hlavního textu této práce, ale také jiných prací a odborných článků s podobným tématem. Dále je krátce zmíněna historie a současný trend akcelerování částicových systémů pomocí GPU grafických karet.

V druhé části je navržen model jednoduchého částicového systému, který je rozdělen na dva na sobě nezávislé celky. Jeden reprezentuje simulační a druhý vykreslovací část částicového systému. Při návrhu byl kladen důraz na rozšiřitelnost a především na zredukování zbytečných a opakujících se operací či volání metod. Je také stručně naznačeno, jak jsem tento návrh implementoval.

Výsledkem této implementace je knihovna, kterou lze použít v jakémkoli jiném projektu, který využívá balík knihoven DirectX 9. Pro tyto případy je na přiloženém CD uložena automaticky vytvořená dokumentace této knihovny, která by měla postačit k porozumění začlenění do jiného projektu.

Možné rozšíření projektu vidím v přenesení veškeré simulace chování částic do GPU grafického adaptéru. Využití textur a pixel shaderů pro uchovávání stavů částic a jejich integrace v čase. Dalším zajímavým rozšířením by bylo vytvořit nadstandardní typy emitorů částic, například emitor, který bude generovat částice na povrchu nepravidelného objektu (mesh) vytvořeného některým z profesionálních nástrojů pro práci s 3D grafikou. Některá další možná rozšíření: Detekce kolize částice a nepravidelných objektů, kolize jednotlivých částic a reakce na ně, nové typy uzlů, například takové, který by využívaly tzv. *metaballs*, viz [11] atd.

Literatura

[1] Kolektiv autorů: *Grafika* [online]. Aktualizováno 2009-04 [cit. 2009-04-07]. Dostupné na URL: <http://cs.wikipedia.org/wiki/Grafika>

[2] Kolektiv autorů: *Počítačová grafika* [online] Aktualizováno 2009-04 [cit. 2009-04-07]. Dostupné na URL: < http://cs.wikipedia.org/wiki/Počítačová_grafika>

[3] G. Scott Owen: *Particle Systems* [online]. Aktualizováno 2000-02 [cit. 2009-04-09]. Dostupné na URL: < http://www.siggraph.org/education/materials/HyperGraph/animation/particle.htm>

[4] Allen Martin: *Particle Systems* [online]. [cit. 2009-04-09]. Dostupné na URL: < http://web.cs.wpi.edu/~matt/courses/cs563/talks/psys.html>

[5] Kolektiv autorů: *Vektor* [online]. Aktualizováno 2009-04 [cit. 2009-04-07]. Dostupné na URL: < http://cs.wikipedia.org/wiki/Vektor>

[6] Jakub Vojáček: *Analytická geometrie - Skalární součin* [online]. Aktualizováno 2008-05 [cit. 2009-04-12]. Dostupné na URL: < http://maths.cz/clanky/analyticka-geometrie-skalarni-soucin.html>

[7] Kolektiv autorů: *Microsoft Developer Network - Volume Texture Resources* [online]. [cit. 2009-04-12]. Dostupné na URL: < http://msdn.microsoft.com/en-us/library/bb206344(VS.85).aspx>

[8] Kolektiv autorů: *Microsoft Developer Network - Texture Filtering* [online]. [cit. 2009-04-13]. Dostupné na URL: < http://msdn.microsoft.com/en-us/library/bb206250.aspx>

[9] Akenine-Möller, T., Haines, E.: *Billboarding* [online]. Aktualizováno 2002-09 [cit. 2009-04-13]. Dostupné na URL: < http://www.flipcode.com/archives/Billboarding-Excerpt_From_iReal-Time_Renderingi_2E.shtml>

[10] Latta, L.: *Building a Million Particle System* [online]. Aktualizováno 2007-06 [cit. 2009-04-15]. Dostupné na URL: < http://2ld.de/gdc2004/MegaParticlesPaper.pdf>

[11] Shen, J., Thalmann, D.: *Interactive shape design using metaballs and splines* [online]. [cit. 2009-04-18]. Dostupné na URL: < http://ligwww.epfl.ch/~thalmann/papers.dir/IS95.Metaballs.pdf>

Seznam příloh